Paleti Krishnasai
CED18I039

Operating Systems - Assignment 6 - THREADS

1. Generate Armstrong number generation within a range.

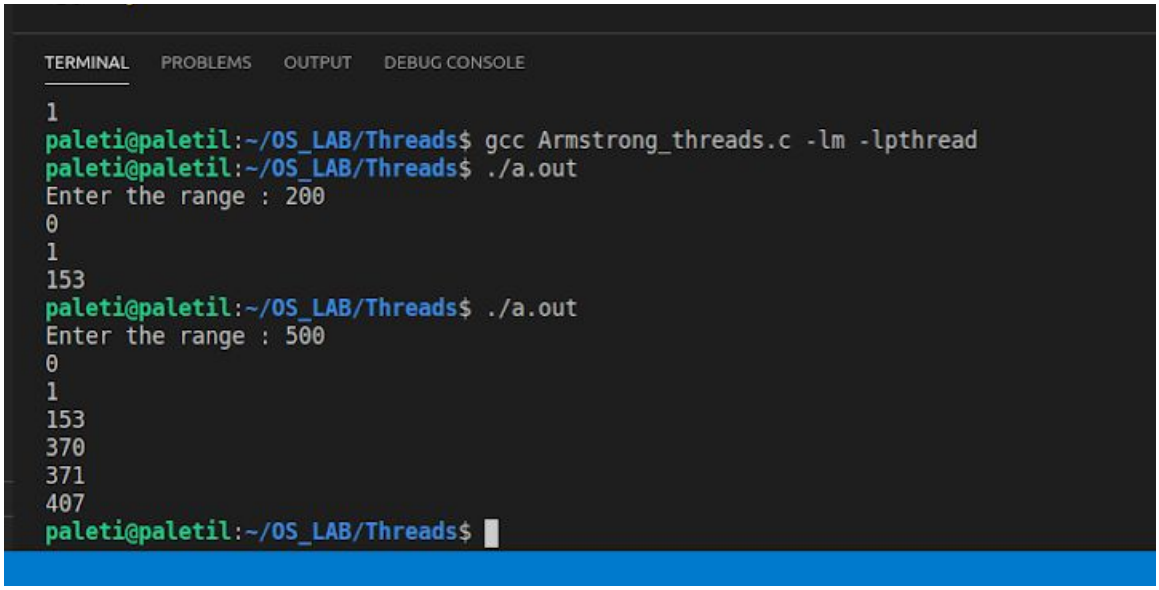
Code : Extension of fork() code to threads.

```c
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



void *child(void *param)

{   int temporary =*((int*) param);

   int num, originalNum, rem, result = 0;


   originalNum = temporary;

   num = temporary;

   while (originalNum != 0)

   {

       // rem contains the last digit
```

```
        rem = originalNum % 10;


        result += rem * rem * rem;


        // removing last digit from the orignal number
        originalNum /= 10;
    }
    if (result == num)
    printf("%d \n", num);
    pthread_exit(NULL);
}


int main()
{
    int start, end ;
    int range;
    printf("Enter the range : ");
    scanf("%d",&end);
    range = end + 1;


    pthread_t tid[range] ;
    for (int i = 0; i <= end; i++)
    {
        pthread_create(&tid[i],NULL,child,&i);
        pthread_join(tid[i], NULL);
    }
```

```
        return 0;
}
```

Output:



Paleti Krishnasai
CED18I039

2. Ascending Order sort and Descending order sort.

Code: Parallelized version where ascending sort and descending sort are done on distinct user threads.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



#define MAX 100



int a[MAX],d[MAX];

int size;



void *ascending(void *param)

{

   for (int i = 0; i < size; i++)

   {

       int max_i = 0;
```

```
        for (int j = 0; j < size - i; j++)

        {

            if (a[j] > a[max_i])

            {

                max_i = j;

            }

        }



        int temp = a[max_i];

        a[max_i] = a[size - i - 1];

        a[size - i - 1] = temp;

    }

    printf("\nAscending order: ");



    for (int i = 0; i < size; i++)

        printf("%d ", a[i]);



    printf("\n");

    pthread_exit(NULL);

}



void *descending(void *arg)
```

Paleti Krishnasai
CED18I039

```
{
   for (int i = 0; i < size; i++)
   {
       int max_i = i;
       for (int j = i; j < size; j++)
       {
           if (d[j] > d[max_i])
           {
               max_i = j;
           }
       }
       int temp = d[max_i];
       d[max_i] = d[i];
       d[i] = temp;
   }
   printf("\nDescending order: ");
   for (int i = 0; i < size; i++)
       printf("%d ", d[i]);


   printf("\n");
   pthread_exit(NULL);
}
```

Paleti Krishnasai
CED18I039

```
int main()

{



       printf("Enter size of array: ");

       scanf("%d", &size);

       printf("Enter the array: ");

       for (int i = 0; i < size; i++)

       {

           scanf("%d", &a[i]);

           d[i] = a[i];

       }

       pthread_t tid[2];

       pthread_create(&tid[0],NULL,ascending,NULL);

       pthread_join(tid[0], NULL);

       pthread_create(&tid[1],NULL,descending,NULL);

       pthread_join(tid[1],NULL);



   return 0;

}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter size of array: 5
Enter the array: 5 4 6 7 8

Ascending order: 4 5 6 7 8

Descending order: 8 7 6 5 4
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter size of array: 6
Enter the array: 5 3 8 2 7 9

Ascending order: 2 3 5 7 8 9

Descending order: 9 8 7 5 3 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter size of array: 7
Enter the array: 1 2 3 9 8 7 6

Ascending order: 1 2 3 6 7 8 9

Descending order: 9 8 7 6 3 2 1
paleti@paletil:~/OS_LAB/Threads$
```

Paleti Krishnasai
CED18I039

3. Implement a multithreaded version of binary search. By default, you can implement a search for the first occurrence and later extend to support multiple occurrence (duplicated elements search as well)

Code: Extension of fork() code to threads .

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



int arr[100];

int pos = -1;

struct data

{

   int beg;

   int end;

   int x;

};

void *BinarySearchMultiple(void *d)
```

```
{
   struct data *dm = d;
   int beg = dm->beg;
   int end = dm->end;
   int x = dm->x;



   if (beg < end)
   {
       int mid = (beg + end) / 2;


       if (arr[mid] == x)
       {
           pos = mid;
       }
       else
       {
           pthread_t tid;
           pthread_attr_t attr;
           pthread_attr_init(&attr);


           struct data d;
```

```
            d.x = x;



            if (x < arr[mid])

            {

                d.beg = beg;

                d.end = mid;

            }

            else if (x > arr[mid])

            {

                d.beg = mid + 1;

                d.end = end;

            }



            pthread_create(&tid, &attr, BinarySearchMultiple, &d);



            pthread_join(tid, NULL);

        }

    }

    pthread_exit(0);

}



void Display(int x)
```

```
{
   printf("'%d' found at indices: ", x);
   int lend = 0;
   while (lend >= 0)
   {
       if (arr[pos - lend] == x)
       {
           printf("%d ", pos - lend);
           lend++;
       }
       else
           lend = -1;
   }


   int rend = 1;
   while (rend >= 1)
   {
       if (arr[pos + rend] == x)
       {
           printf("%d ", pos + rend);
           rend++;
       }
```

```
        else

            rend = -1;

    }

}



int main(int argc, char *argv[])

{



    if (argc < 3)

        printf("Usage: ./a.out <key> <list>\n");

    else

    {

        int n = argc - 2;



        for (int i = 0; i < n; i++)

        {

            arr[i] = atoi(argv[2 + i]);

        }



        // Sort the array



        for(int i=0;i<n-1;i++) {
```

```
            for(int j=0;j<n-i-1;j++) {
                if(arr[j] > arr[j+1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
        printf("Sorted list : ");
        for(int i=0;i<n;i++) printf("%d ", arr[i]);
        printf("\n");


        struct data d[4];
        d[0].beg = 0;
        d[0].end = n / 4 - 1;
        d[0].x = atoi(argv[1]);
        d[1].beg = n / 4;
        d[1].end = n / 2 - 1;
        d[1].x = atoi(argv[1]);
        d[2].beg = n / 2;
        d[2].end = 3 * n / 4 - 1;
```

```
    d[2].x = atoi(argv[1]);

    d[3].beg = 3 * n / 4;

    d[3].end = n - 1;

    d[3].x = atoi(argv[1]);



    pthread_t tid[4];

    pthread_attr_t attr;

    pthread_attr_init(&attr);



    for (int i = 0; i < 4; i++)

    {

        pthread_create(&tid[i], &attr, BinarySearchMultiple, &d[i]);

    }



    for (int i = 0; i < 4; i++)

    {

        pthread_join(tid[i], NULL);

    }



    if (pos > -1)

        Display(atoi(argv[1]));

    else
```

```
            printf("\n%d not found\n", atoi(argv[1]));



        printf("\n");

    }



}
```

Output:

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out 6 7 8 9 4 6 4 5 2 6
Sorted list : 2 4 4 5 6 6 7 8 9
'6' found at indices: 4 5
paleti@paletil:~/OS_LAB/Threads$ ./a.out 4 7 8 9 4 6 4 5 2 6
Sorted list : 2 4 4 5 6 6 7 8 9
'4' found at indices: 2 1
paleti@paletil:~/OS_LAB/Threads$
```

```
paleti@paletil:~/OS_LAB/Threads$ gcc BSThreaded.c -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 6 8 9 5 9 5 2 6 5 3
Sorted list : 2 3 5 5 5 6 6 8 9 9
'5' found at indices: 3 2 4
paleti@paletil:~/OS_LAB/Threads$ ./a.out 6 6 8 9 5 9 5 2 6 5 3
Sorted list : 2 3 5 5 5 6 6 8 9 9
'6' found at indices: 5 6
```

Paleti Krishnasai
CED18I039

4.Generation of Prime Numbers upto a limit supplied as Command Line Parameter.

Code: Primality check for each number within the range , where each number is checked on a distinct user thread.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



void *primecheck(void *param )

{

   int n = *((int *)param);

   int j,flag=1;

   if (n <= 1)

       flag = 0;

   else if (n == 2)
```

```
        flag = 1;

    else

    {

        for(j=2;j<=sqrt(n);j++)

        {

            if(n%j == 0)

            {

                flag=0;

                break;

            }

        }

    }

    if(flag == 1)

    printf("%d\n",n);

}



int main(int argc,const char *argv[])

{

    if(argc!=2)

    printf("Usage: ./a.out num(limit of prime numbers to be
generated)\n");

    else
```

Paleti Krishnasai
CED18I039

```
    {

        int count = atoi(argv[1]);

        pthread_t tid[count*2];

        pthread_attr_t attr;

        pthread_attr_init(&attr);



        for (int i=1;i<=count;i++)

        {

            pthread_create(&tid[i],&attr,primecheck,&i);

            pthread_join(tid[i],NULL);

        }



    }

    return 0;

}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/Threads$ gcc primegen.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ gcc primegen.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Usage: ./a.out num(limit of prime numbers to be generated)
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5
2
3
5
paleti@paletil:~/OS_LAB/Threads$ ./a.out 20
2
3
5
7
11
13
17
19
paleti@paletil:~/OS_LAB/Threads$ ./a.out 50
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
paleti@paletil:~/OS_LAB/Threads$
```

Paleti Krishnasai
CED18I039

5. Computation of Mean, Median, Mode for an array of integers.

Code: A parallelized computation of mean , median and mode , running on distinct user threads.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>

#define MAX 100

int list[MAX];

int size;

int cmpfunc(const void *a, const void *b)

{

   return (*(int *)a - *(int *)b);

}

void *mean(void *param)

{
```

```
    int temp[size];
    for (int i=0;i<size;i++)
    {
        temp[i] = list[i];
    }
    float mean;
    for (int i = 0; i < size; i++)
    {
        mean = mean + temp[i];
    }
    mean = mean / size;
    printf("\nMean : %f\n",mean);
    pthread_exit(NULL);
}


void *median(void *param)
{
    int temp[size];
    for (int i = 0; i < size; i++)
    {
        temp[i] = list[i];
    }
```

```
   qsort(temp, size, sizeof(int), cmpfunc);

   int median;

   if(size%2==1)

   median = temp[(size - 1)/2];

   else median = (temp[size/2] + temp[size/2 - 1]) / 2 ;

   printf("\nMedian : %d\n",median);

   pthread_exit(NULL);

}



void *mode(void *param)

{



   int temp[size];

   for (int i = 0; i < size; i++)

   {

       temp[i] = list[i];

   }

   int mode;

   int maxCount;

   for (int i = 0; i < size; ++i)

   {

       int count = 0;
```

```
        for (int j = 0; j < size; ++j)
        {
            if (temp[j] == temp[i])
                ++count;
        }

        if (count > mode)
        {
            maxCount = count;
            mode = temp[i];
        }
    }
    printf("\nMode : %d\n",mode);
    pthread_exit(NULL);
}

int main(int argc, char const *argv[])
{
    if(argc<2)
    printf("Usage: ./a.out <list of numbers>\n");
    size = argc - 1;
```

Paleti Krishnasai
CED18I039

```
    for (int i = 0; i < size; i++)

    {

        list[i] = atoi(argv[i+1]);

    }

    pthread_t tid[3];

    pthread_attr_t attr;

    pthread_attr_init (&attr);



    pthread_create(&tid[0],&attr,mean,NULL);

    pthread_create(&tid[1], &attr, median, NULL);

    pthread_create(&tid[2], &attr, mode, NULL);

    pthread_join(tid[0], NULL);

    pthread_join(tid[1], NULL);

    pthread_join(tid[2], NULL);

    return 0;

}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/Threads$ gcc MMM.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 6

Mean : 4.000000

Median : 4

Mode : 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2

Mean : 3.200000

Median : 3

Mode : 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9

Median : 4

Mean : 4.200000

Mode : 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9

Mean : 4.200000

Median : 4

Mode : 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9

Mean : 4.200000

Median : 4

Mode : 2
```

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9

Mean : 4.200000

Median : 4

Mode : 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9 10 22 36

Mode : 2

Mean : 8.461538

Median : 5
paleti@paletil:~/OS_LAB/Threads$ ./a.out 2 3 4 5 2 5 4 2 6 9 10 22 36

Mean : 8.461538

Median : 5

Mode : 2
paleti@paletil:~/OS_LAB/Threads$
```

Paleti Krishnasai
CED18I039

6. Implement Merge Sort and Quick Sort in a multithreaded fashion.

Code: Extension of fork() code to threads.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#define MAX 100
int qarray[MAX];
int marray[MAX];

struct data
{
   int beg;
   int end;
};

int Partition(int beg, int end)
{
   int i = beg, j = end;
   int p;
   p = beg;

   int val = p;
   while (i < j)
   {
       while (qarray[p] >= qarray[i] && i < end)
       {
           i++;
       }
```

```
        while (qarray[p] < qarray[j] && j > beg)
        {
            if (j == p + 1 && i <= p)
                j = p - 1;
            //if(j!=beg)j--;
            else
                j--;
        }
        if (i < j)
        {
            int temp = qarray[i];
            qarray[i] = qarray[j];
            qarray[j] = temp;
        }
    }
    int temp = qarray[p];
    qarray[p] = qarray[j];
    qarray[j] = temp;
    val = j;
    return val;
}

void *QuickSort(void *arg)
{
    struct data *temp = arg;
    int beg = temp->beg;
    int end = temp->end;

    if (beg < end)
    {
        int j = Partition(beg, end);

        struct data left;
        left.beg = beg;
        left.end = j - 1;
```

```
        struct data right;
        right.beg = j + 1;
        right.end = end;

        pthread_t tid[2];
        pthread_attr_t attr;
        pthread_attr_init(&attr);

        pthread_create(&tid[0], &attr, QuickSort, &left);
        pthread_create(&tid[1], &attr, QuickSort, &right);

        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);
    }

    pthread_exit(0);
}

void MergeArray(int beg, int mid, int end)
{
    int n1 = mid - beg + 1;
    int n2 = end - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = marray[beg + i];
    for (int j = 0; j < n2; j++)
        R[j] = marray[mid + 1 + j];

    int i = 0, j = 0, k = beg;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
```

```
            marray[k] = L[i];
            i++;
        }
        else
        {
            marray[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        marray[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        marray[k] = R[j];
        j++;
        k++;
    }
}

void *MergeSort(void *arg1)
{
    struct data *temp1 = arg1;
    int beg = temp1->beg;
    int end = temp1->end;

    if (beg < end)
    {
        int mid = (beg + end) / 2;

        struct data left;
```

```
        left.beg = beg;
        left.end = mid;
        struct data right;
        right.beg = mid + 1;
        right.end = end;

        pthread_t tid[2];
        pthread_attr_t attr;
        pthread_attr_init(&attr);

        pthread_create(&tid[0], &attr, MergeSort, &left);
        pthread_create(&tid[1], &attr, MergeSort, &right);

        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);

        MergeArray(beg, mid, end);
    }
    pthread_exit(0);
}

int main(int argc, char *argv[])
{

    if (argc < 2)
        printf("Usage: ./a.out <list of numbers>\n");
    else
    {
        int size = argc - 1;

        for (int i = 0; i < size; i++)
        {
            qarray[i] = atoi(argv[1 + i]);
            marray[i] = atoi(argv[1 + i]);
        }
```

```
        struct data param;
        param.beg = 0;
        param.end = size - 1;

        pthread_t tid[2];
        pthread_attr_t attr;
        pthread_attr_init(&attr);

        pthread_create(&tid[0], &attr, MergeSort, &param);
        pthread_create(&tid[1], &attr, QuickSort, &param);

        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);

        printf("Merge Sort: ");

        for (int i = 0; i < size; i++)
            printf("%d ", qarray[i]);

        printf("\n");

        printf("Quick Sort: ");

        for (int i = 0; i < size; i++)
            printf("%d ", marray[i]);

        printf("\n");
    }
    return 0;
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/Threads$ gcc mergequick.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6
Merge Sorted: 1 2 3 4 5 6 8 9
Quick Sorted: 1 2 3 4 5 6 8 9
paleti@paletil:~/OS_LAB/Threads$ gcc mergequick.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6
Merge Sort: 1 2 3 4 5 6 8 9
Quick Sort: 1 2 3 4 5 6 8 9
paleti@paletil:~/OS_LAB/Threads$ gcc mergequick.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6
Merge Sort: 1 2 3 4 5 6 8 9
Quick Sort: 1 2 3 4 5 6 8 9
paleti@paletil:~/OS_LAB/Threads$ gcc mergequick.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6
Merge Sort: 1 2 3 4 5 6 8 9
Quick Sort: 1 2 3 4 5 6 8 9
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6 32 56 99 2002 24 12
Merge Sort: 1 2 3 4 5 6 8 9 12 24 32 56 99 2002
Quick Sort: 1 2 3 4 5 6 8 9 12 24 32 56 99 2002
paleti@paletil:~/OS_LAB/Threads$ gcc mergequick.c -lm -lpthread
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6 32 56 99 2002 24 12
Merge Sort: 1 2 3 4 5 6 8 9 12 24 32 56 99 2002
Quick Sort: 1 2 3 4 5 6 8 9 12 24 32 56 99 2002
paleti@paletil:~/OS_LAB/Threads$ ./a.out 5 2 3 4 9 8 1 6 32 56 99 2002 24 12 56 105 1
001 1024 1020
Merge Sort: 1 2 3 4 5 6 8 9 12 24 32 56 56 99 105 1001 1020 1024 2002
Quick Sort: 1 2 3 4 5 6 8 9 12 24 32 56 56 99 105 1001 1020 1024 2002
paleti@paletil:~/OS_LAB/Threads$ 
```

Paleti Krishnasai
CED18I039

7. Estimation of PI Value using Monte carlo simulation technique (refer the internet for the method..) using threads.

Code: Reference

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



int circle_points = 0;

int square_points = 0;

double rand_x,rand_y,origin_dist;



int interval;



void *runner(void *p)

{

   rand_x  = (double)(rand() % (interval + 1)) / interval;

   rand_y =  (double)(rand() % (interval + 1)) / interval;
```

```
    origin_dist = (rand_x * rand_x) + (rand_y * rand_y);

    if(origin_dist <= 1)

        circle_points++;

    square_points++;



    pthread_exit(NULL);

}



int main(int argc,char *argv[])

{

    if(argc != 2)

    printf("Usage: ./a.out <interval>\n");

    else

    {

        interval = atoi(argv[1]);

        pthread_t tid[interval*interval];

        pthread_attr_t attr;

        pthread_attr_init (&attr);

        srand(time(NULL));



        for(int i = 0;i<(interval*interval);i++)

        pthread_create(&tid[i],&attr,runner,NULL);
```

```
        for(int i = 0;i<(interval*interval);i++)

        pthread_join(tid[i],NULL);



        double pi = (double)(4 * circle_points) / square_points;

        printf("\nFinal Estimation of Pi = %f,\n", pi);

    }

    printf("\n");

    return 0;


}
```

Output:

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out 9

Final Estimation of Pi = 3.407407,

paleti@paletil:~/OS_LAB/Threads$ ./a.out 100

Final Estimation of Pi = 3.123200,
```

Paleti Krishnasai
CED18I039

Optional:
8. Computation of a Matrix Inverse using Determinant, Cofactor threads, etc.

Code:

Reference . Run cofactor function with threads , where each cell value is run as a thread.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>

#define MAX 100

int N;

int matrix[MAX][MAX];

int adjoint[MAX][MAX];

float inverse[MAX][MAX];



struct cell

{
```

```
   int i;
   int j;


};
/* Function to get cofactor of matrix[p][q] in temp[][]. N is current
dimension of matrix[][]*/
void cofactor(int mat[MAX][MAX], int temp[MAX][MAX],int p,int q,int n)
{
   int i=0,j=0;
   for(int row=0 ; row<n ; row++)
   {
       for(int col=0 ; col<n ; col++)
       {
           if(row!=p && col!=q)
           {
               temp[i][j++] = mat[row][col];
               if (j==n-1)
               {
                   j=0;
                   i++;
               }
           }
```

```
        }

    }

}



int determinant(int mat[MAX][MAX],int n)

{

    int D=0;

    if(n==1)

    return mat[0][0];



    int temp[N][N]; // To store cofactors

    int sign = 1;   // To store sign multiplier



    // Iterate for each element of first row

    for(int f=0 ; f<n ; f++)

    {

        cofactor(mat,temp,0,f,n);

        D += sign * mat[0][f] * determinant(temp,n-1);

        sign = -sign;

    }

    return D;

}
```

Paleti Krishnasai
CED18I039

```
void *runner(void *param)

{

   int temp[N][N];

   int sign = 1;



   struct cell *data = param;

   cofactor(matrix,temp,data->i,data->j,N);

   sign = ((data->i + data->j)%2 == 0) ? 1 :-1;

   adjoint[data->j][data->i] = sign * (determinant(temp,N-1));

   pthread_exit(NULL);

}



int main()

{

   printf("Enter the order : \n");

   scanf("%d", &N);

   printf("Enter the matrix :\n");

   for (int i=0;i<N;i++)

   {

       for (int j=0;j<N;j++)

       {
```

```
            scanf("%d",&matrix[i][j]);

        }

    }

    printf("\nMatrix:\n");

    for(int i=0;i<N;i++)

    {

        for(int j=0;j<N;j++)

        printf("%d ",matrix[i][j]);

        printf("\n");

    }

    pthread_t tid[N*N];

    pthread_attr_t attr;

    pthread_attr_init(&attr);



    if (N == 1)

    {

        adjoint[0][0] = 1;

    }

    else

    {

        int k = 0;

        for (int i = 0; i < N; i++)
```

```
        {
            for (int j = 0; j < N; j++)
            {
                struct cell *d = (struct cell *)malloc(sizeof(struct
cell));


                d->i = i;
                d->j = j;


                pthread_create(&tid[k], &attr, runner, d);
                k++;
            }
        }
        for (int i = 0; i < k; i++)
            pthread_join(tid[i], NULL);
    }
    printf("\nAdjoint:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d ", adjoint[i][j]);
        printf("\n");
```

```
    }

    int det = determinant(matrix, N);

    if (det == 0)

    {

        printf("Inverse doesn't exist\n");

        return 0;

    }

    for (int i = 0; i < N; i++)

        for (int j = 0; j < N; j++)

            inverse[i][j] = adjoint[i][j] / ((float)det);



    printf("Inverse:\n");

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < N; j++)

            printf("%f ", inverse[i][j]);

        printf("\n");

    }

    printf("\n");

    return 0;

}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter the order :
4
Enter the matrix :
5 -2 2 7
1 0 0 3
-3 1 5 0
3 -1 -9 4

Matrix:
5 -2 2 7
1 0 0 3
-3 1 5 0
3 -1 -9 4

Adjoint:
-12 76 -60 -36
-56 208 -82 -58
4 4 -2 -10
4 4 20 12
Inverse:
-0.136364 0.863636 -0.681818 -0.409091
-0.636364 2.363636 -0.931818 -0.659091
0.045455 0.045455 -0.022727 -0.113636
0.045455 0.045455 0.227273 0.136364
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter the order :
3
Enter the matrix :
1 1 2
2 1 1
1 1 1

Matrix:
1 1 2
2 1 1
1 1 1

Adjoint:
0 1 -1
-1 -1 3
1 0 -1
Inverse:
0.000000 1.000000 -1.000000
-1.000000 -1.000000 3.000000
1.000000 0.000000 -1.000000
```

Paleti Krishnasai
CED18I039

9. Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a multithreaded fashion to contribute a faster version of fib series generation.

Code: extension of the fork() code to threads .

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <pthread.h>



struct keyvalue

{

   int key;

   int value;

};



int fib(int n)

{

   if(n==0 || n==1)
```

```
    return n;

    else

    return fib(n-1)+fib(n-2);

}



void *runner(void *param)

{

    struct keyvalue *temporary = (struct keyvalue*)param;

    temporary->value = fib(temporary->key);

    pthread_exit(NULL);

}



int main()

{

    int range;

    printf("enter the number of fibonacci numbers to generate : \n");

    scanf("%d", &range);

    struct keyvalue *generate=(struct keyvalue*)malloc(range*sizeof(struct
keyvalue));



    pthread_t tid[range];

    printf("Fibonacci Series of %d terms\n",range);
```

```
    for (int i=0;i<range;i++)

    {

        generate[i].key =i;

        pthread_create(&tid[i],NULL,runner,&generate[i]);

        pthread_join(tid[i],NULL);

    }

    for(int i=0;i<range;i++)

    {

        printf("%d\n",generate[i].value);

    }



    return 0;

}
```

Output:

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
enter the number of fibonacci numbers to generate :
5
Fibonacci Series of 5 terms
0
1
1
2
3
```

Paleti Krishnasai
CED18I039

```
enter the number of fibonacci numbers to generate :
40
Fibonacci Series of 40 terms
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
```

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
enter the number of fibonacci numbers to generate :
10
Fibonacci Series of 10 terms
0
1
1
2
3
5
8
13
21
34
paleti@paletil:~/OS_LAB/Threads$
```

Paleti Krishnasai
CED18I039

10. Longest common subsequence generation problem using threads.

Code : Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>

int max(int x, int y)
{
   return (x > y ? x : y);
}

struct data
{
   char *s1, *s2;
   int *res;
};

void *runner(void *);

void *longestSubsequence(char *s1, char *s2, int *res)
{
   if (s1[0] == '\0' || s2[0] == '\0')
       *res = 0;
   else if (s1[0] == s2[0])
   {
       int *res1 = malloc(sizeof(int));
       longestSubsequence(&s1[1], &s2[1], res1);
       *res = 1 + *res1;
   }
   else
   {
       int *res1 = malloc(sizeof(int));
       int *res2 = malloc(sizeof(int));
```

```
        struct data d1, d2;

        d1.s1 = &s1[1];
        d1.s2 = s2;
        d1.res = res1;

        d2.s1 = s1;
        d2.s2 = &s2[1];
        d2.res = res2;

        pthread_t tid[2];
        pthread_create(&tid[0], NULL, runner, &d1);
        pthread_create(&tid[1], NULL, runner, &d2);

        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);

        *res = max(*res1, *res2);
    }
}

void *runner(void *params)
{
    struct data *d = params;
    longestSubsequence(d->s1, d->s2, d->res);
}

int main()
{
    char str1[100], str2[100];
    printf("Enter the strings\n");
    fgets(str1, 100, stdin);
    str1[strlen(str1) - 1] = '\0';
    fgets(str2, 100, stdin);
    str2[strlen(str2) - 1] = '\0';

    int *res = malloc(sizeof(int));
    longestSubsequence(str1, str2, res);
    printf("\nLength: %d\n", *res);
```

```
}
```

Output :

```
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter the strings
hello
el

Length: 2
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter the strings
operating
rate

Length: 3
paleti@paletil:~/OS_LAB/Threads$ ./a.out
Enter the strings
abcde
ace

Length: 3
paleti@paletil:~/OS_LAB/Threads$
```